Tirgul 10

AVL Trees
Graphs
DFS

sample questions

Q. We have two BSTs, **A** having n nodes and **B** having m nodes, where n ≥ m. We would like to find their intersection (values that appear in both trees) so we decide to perform an inorder traversal of **A** and search for each value in **B**. What is the worst-case running time of our approach? What if we traverse BST **B** and search for the values in **A**?

A. We traverse through all the members of **A** (n members). For each of these members we perform a search in **B**. Since the running time of search is proportional to the height of the tree **B** which is m in the worst case the total

Q. This time **A** and **B** are AVL trees. What is the worst-case running time of our approach? What if we traverse B and search for the values in **A**?

A. We traverse through all the members of **A** (n members). For each of these members we perform a search in **B**. Since the running time of search is proportional to the height of the tree **B** which is O(log(m)) in the worst case ⇒ The total running time is O(n*log(m)).

Traversing through **B** and searching for the members of **A** yields a running time of O(m*log(n)) (which is better if n > m)

Q. Can you think of a better approach?

A. We dump the trees into lists (inorder traversal: O(n) + O(m)) and find the intersection between the lists.

Finding the intersection between two sorted lists is linear in the number of elements in both lists:

```
List listsIntersection(L1,L2) {
  L3 = new List;
  while(L1 != NULL && L2 != NULL) {
        if (L1.data ≤ L2.data) {
              if (L1.data == L2.data) {
                    L3.add(L1.data);
                    L2 = L2.next; }
              L1 = L1.next;}
        else { L2 = L2.next;}}
  return L3;
}
```

in each iteration we advance at least one of the pointers: The total number of iterations ≤ n+m

Q. We have two AVL trees of height h such that all elements in A are smaller than all elements in B, write an efficient algorithm for building an AVL tree C containing all the members of A and B, what is its complexity?

A. Remove the maximum of A (O(h)), use it as a root for C (A is the left subtree, while B is the right subtree)

Q. What if A is higher than B by 2?

A. We can still merge the trees efficiently like in the previous case but we might have to fix C using rotations.

DFS – pseudo code

```
DFS(G)
      //initializing.
      for each vertex u∈V[G] {
            u.color = white;
            u.prev = nil;
      }
      time = 0;
      for each vertex u ∈V[G] {
            if (u.color == white)
                  DFS-VISIT(u)
      }
```

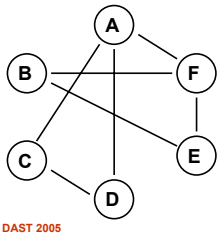
DFS – pseudo code (cont.)

```
DFS-VISIT(u)
     u.color = gray;
     u.d = ++time;
     for each vertex v∈adj[u] {
          if (v.color == white) {
               v.prev = u;
               DFS-VISIT(v);
          }
     }
     u.color = black;
     u.f = ++time;
```

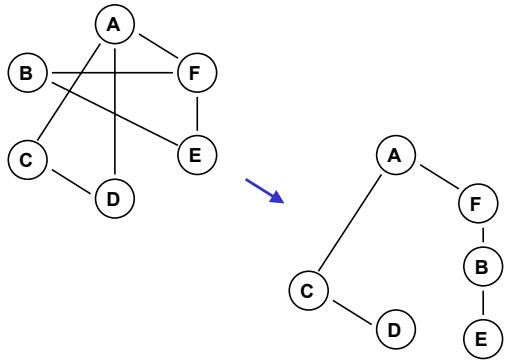

Q. Let G be a graph having |V| vertices and |E| edges
What is the asymptotic time complexity of running DFS on G when G is represented using: (a) an adjacency list? (b) an adjacency matrix? When will the asymptotic time complexity be identical? When will we have the largest difference?

A. The first stage goes through all the vertices (O(|V|) regardless of the representation)
In the second stage we have a constant amount of work for each potential edge, which is O(|E|) in case of an adjacency list and O($|V^2|$) in case of an adjacency matrix.
In full graphs O(|E|) == O($|V^2|$), the sparser the graph, the bigger the difference (in a tree O(|E|) == O(|V|))

Q. Apply DFS to the following graph. Whenever the algorithm has a choice, assume that the node that comes earlier in the alphabet is considered first (like *B* before *E*). (This makes the algorithm deterministic, so you can demonstrate that you understand it fully.) Give the resulting DFS-tree(s) for the graph.



A.



Q. Is it true that in an undirected graph G, if there is a path between two vertices u and v then in the DFS-tree(s) of G, either v is a sibling of u or u is a sibling of v. Proof or give a counter example

A. False (in the previous graph there is a path from C to F yet they are not siblings of one another)

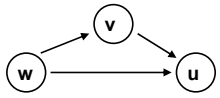
Q. Suggest an algorithm (pseudo-code) for labeling (with labels 1,2,…) the connected components in an undirected graph (all vertices in the same connected component should have the same label). What is it's complexity?

A. This is a very small modification of the DFS algorithm. Instead of coloring each finished node in black we color it with a label. We increase the label for each connected component (outer loop)

➲ the total time complexity is therefore $O(|V+E|)$

Q. In a directed graph G, show how a vertex v might have both incoming and outgoing edges, yet its tree in the DFS forest contains only v.

A.



DFS(u)
DFS(v)
DFS(w)

Q. q4,5 from ex7